CSE633 Boids

Flocking Simulation: by Shaun Cosgrove

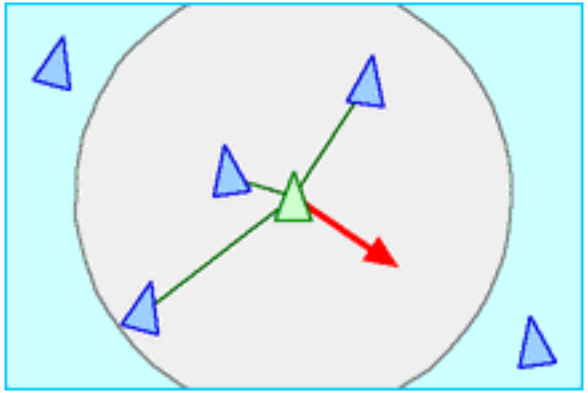
Boids

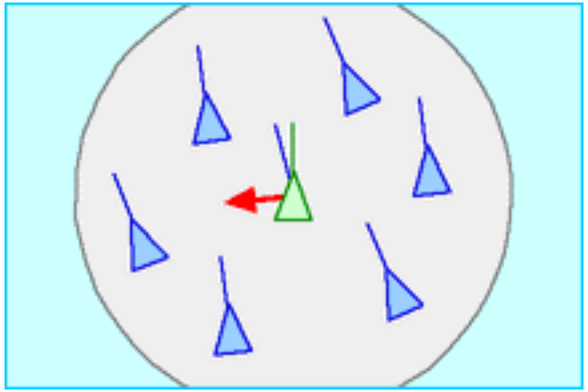
“Boids” is the phonetic spelling of “Birds” when spoken with a New York accent

Flocking Simulation

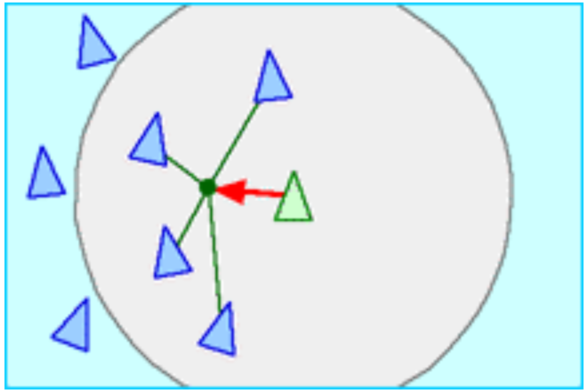
- Originally simulator for the flocking behaviour of birds
- Simple set of rules
  - Emergent Behaviour
    - Complexity arises from the interaction of individual Boids

Boids Rules



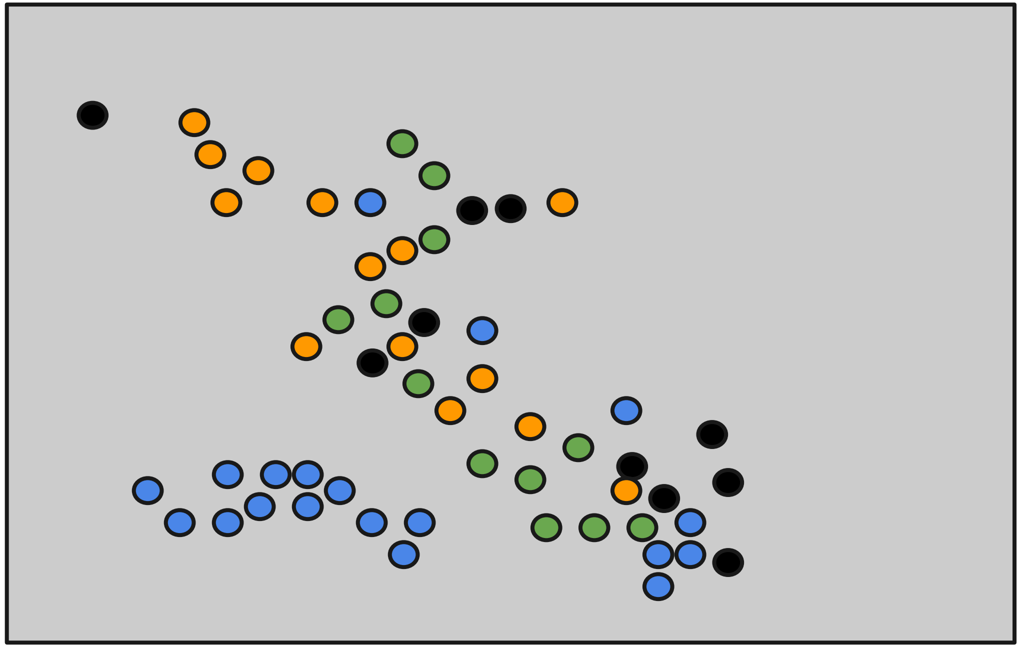


- Separation
- Alignment
- Cohesion



Images Credit http://www.red3d.com/cwr/boids/

Implementation Example

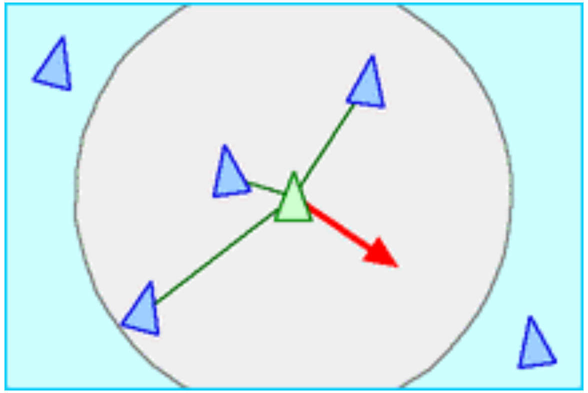


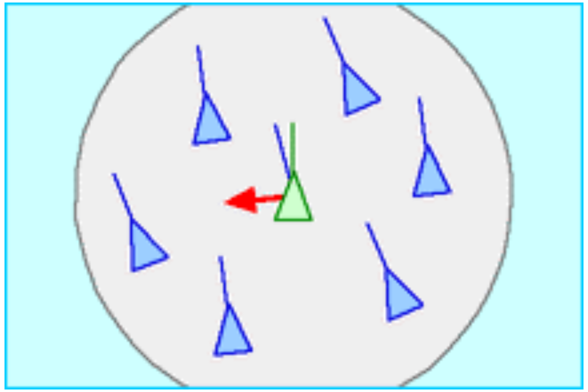
Processor 1	●
Processor 2	🟢
Processor 3	🟠
Processor 4	🔵
….	
….	
….	

CSE633 Boids - Implementation & Analysis

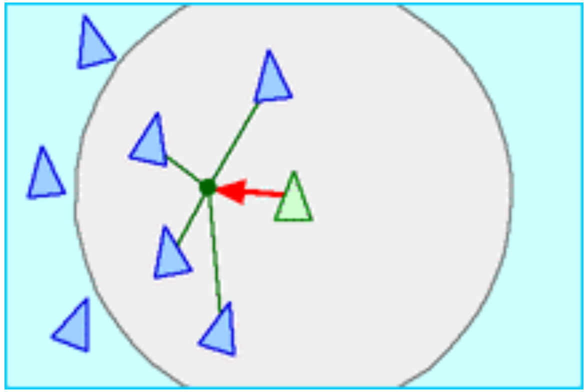
Flocking Simulation: by Shaun Cosgrove

Boids Rules



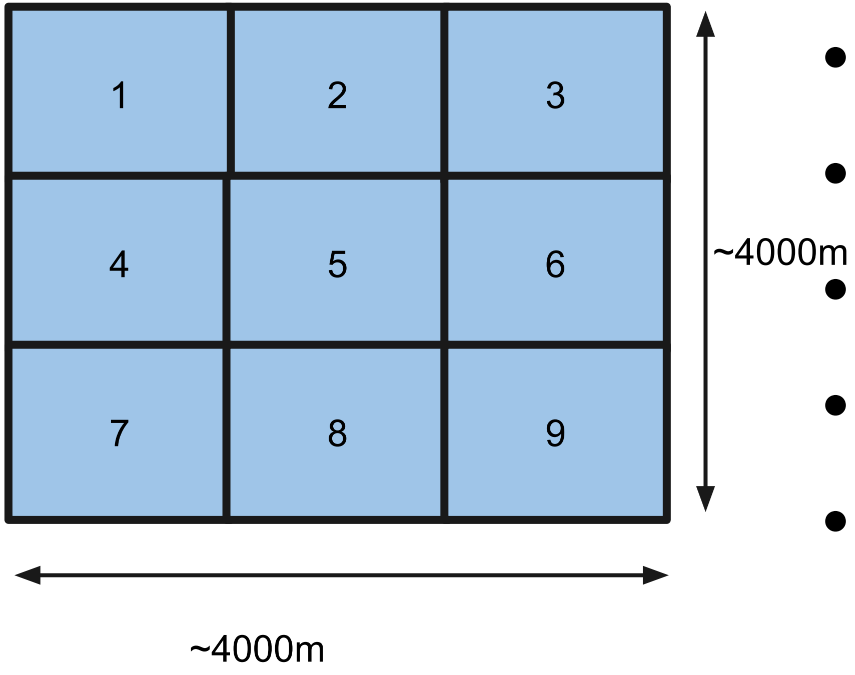


- Separation
- Alignment
- Cohesion



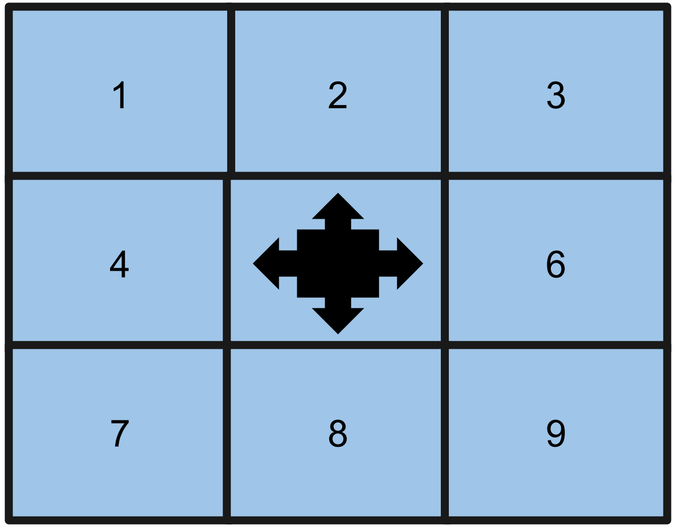
Images Credit http://www.red3d.com/cwr/boids/

Static Spatial Decomposition



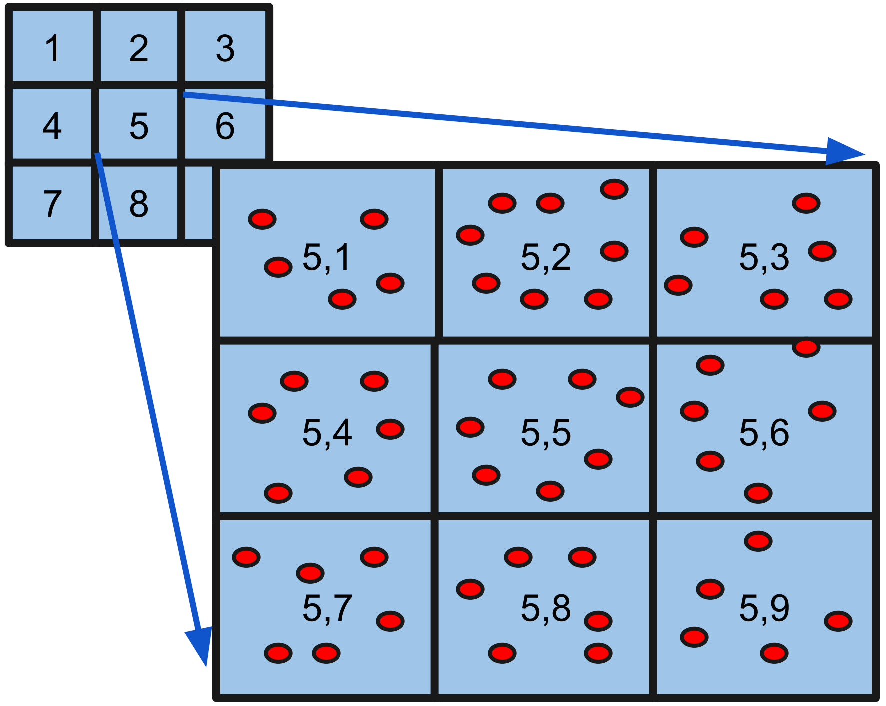
- Divide total area into sub areas
- Assign each sub area to a processor
- Perform computation on each sub area
- Sync data
- Repeat

Shared Borders



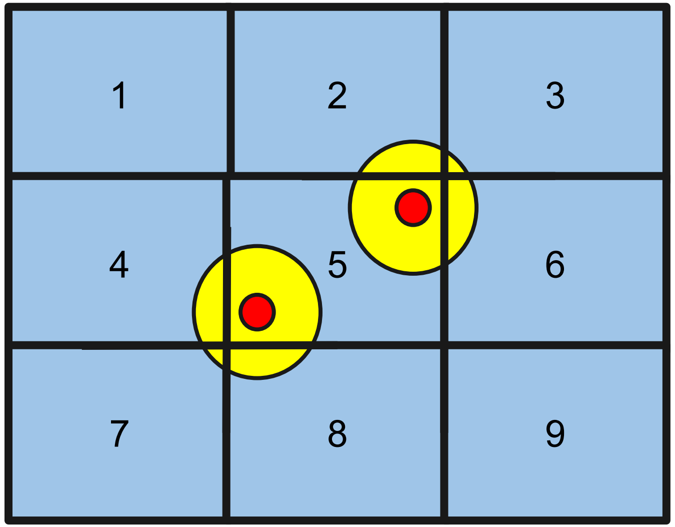
- Computation on any sub areas requires data from every neighbour
- Transmit all boids to / from neighbours?
- Necessary to reduce communication requirements for faster implementation

Sub Area Optimizations



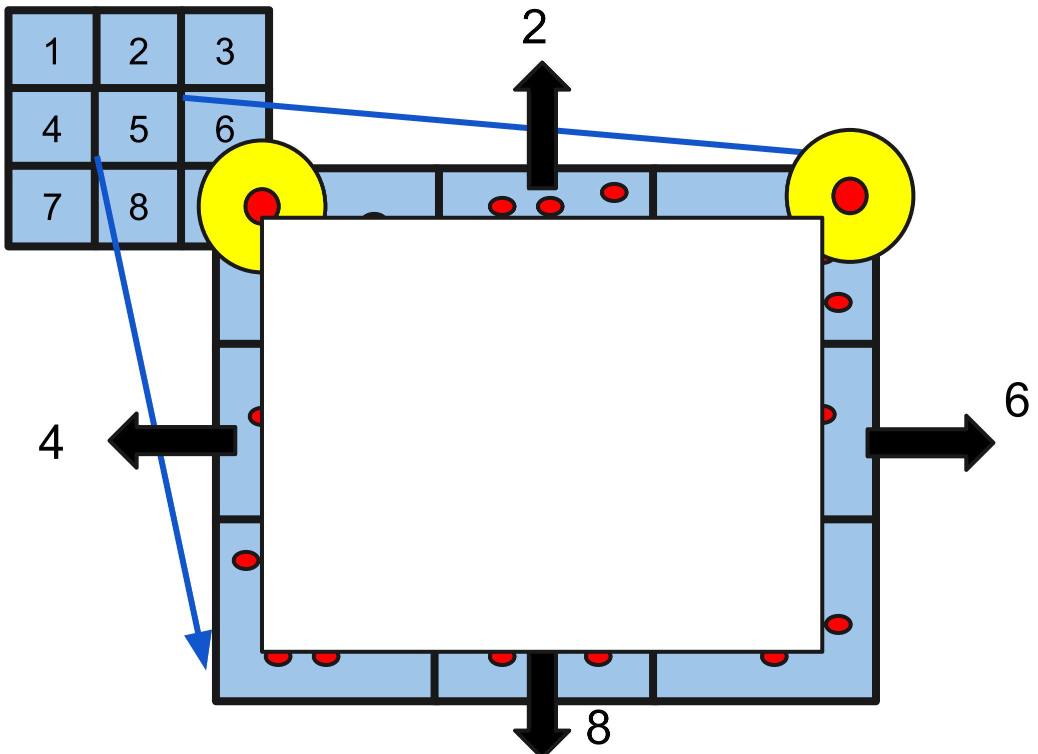
- Binning of Boids
- Each sub area divided into bins
- Boid algorithm performed on binned Boids

Border Communication



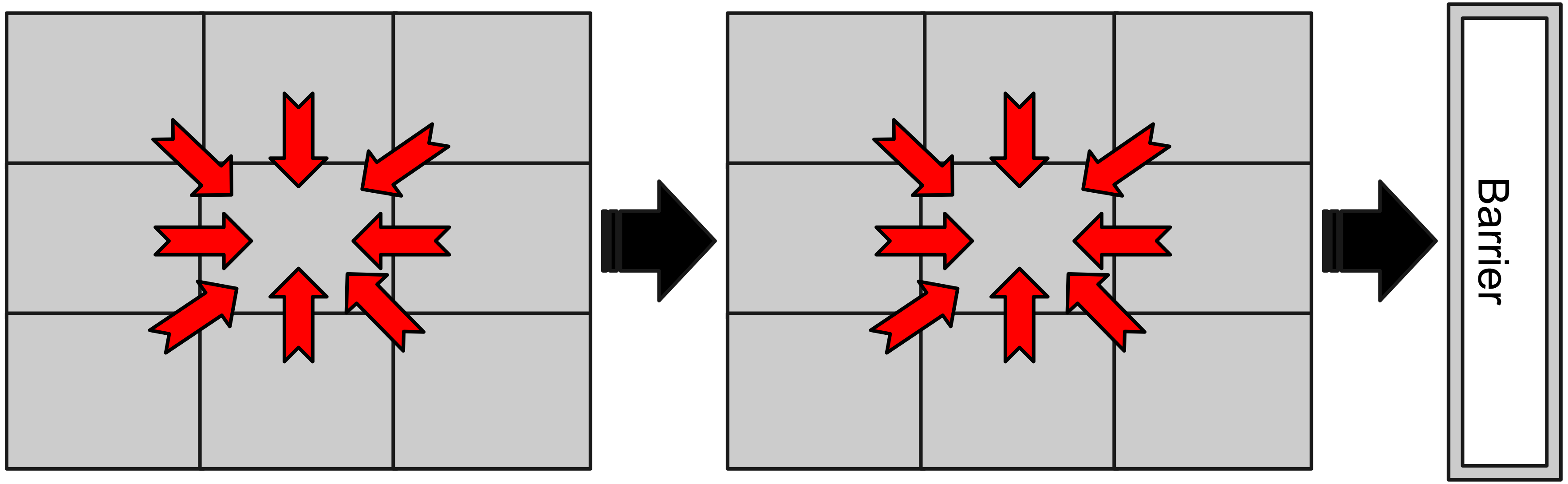
- Boids sphere of influence goes outside local sub area
- Correct Simulation Requires Data from each neighbour
- Make each computation communicate with neighbours for data?
- How much data to get?
- Efficiency very important

Communication Optimization



- Take the border bins
- **TX** and **RX** **all boids** in these bins to their respective neighbours
- **Buffer neighbour Boids** for processing in all sub-areas
- Ensures **correct simulation** in all sub areas

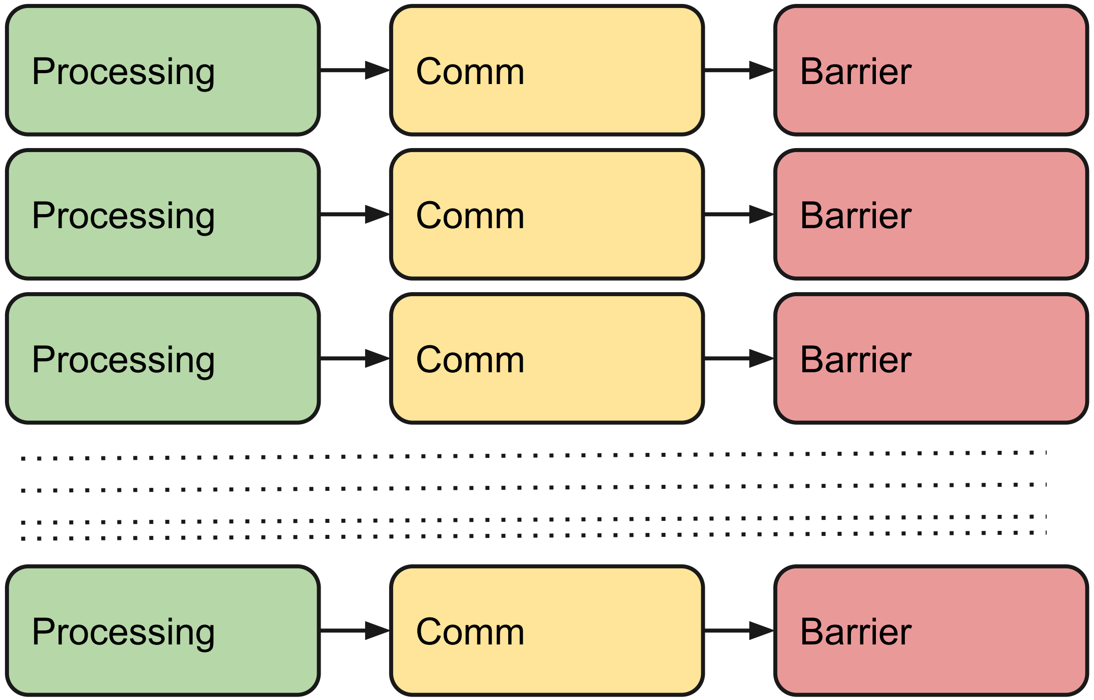
Communication States



Boid Lengths

Boid Lists

Application Sequence

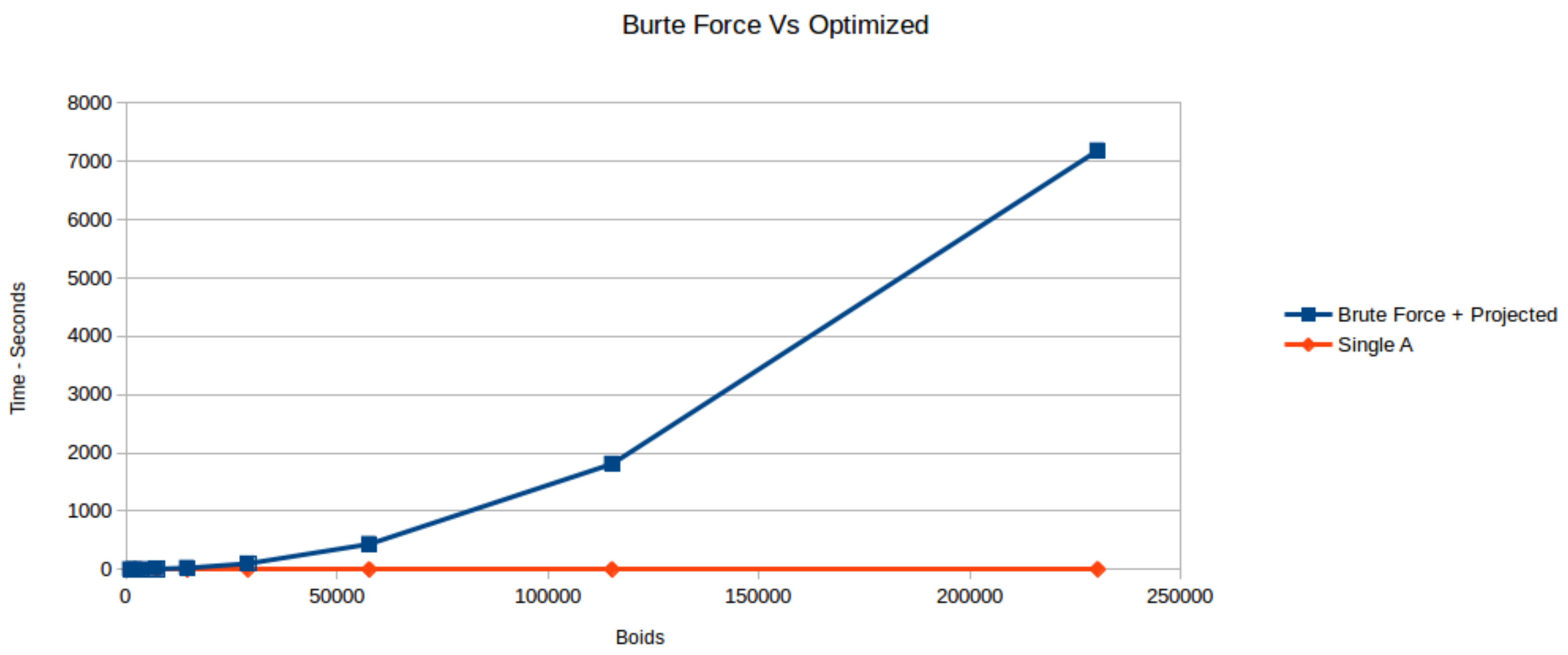


Notes

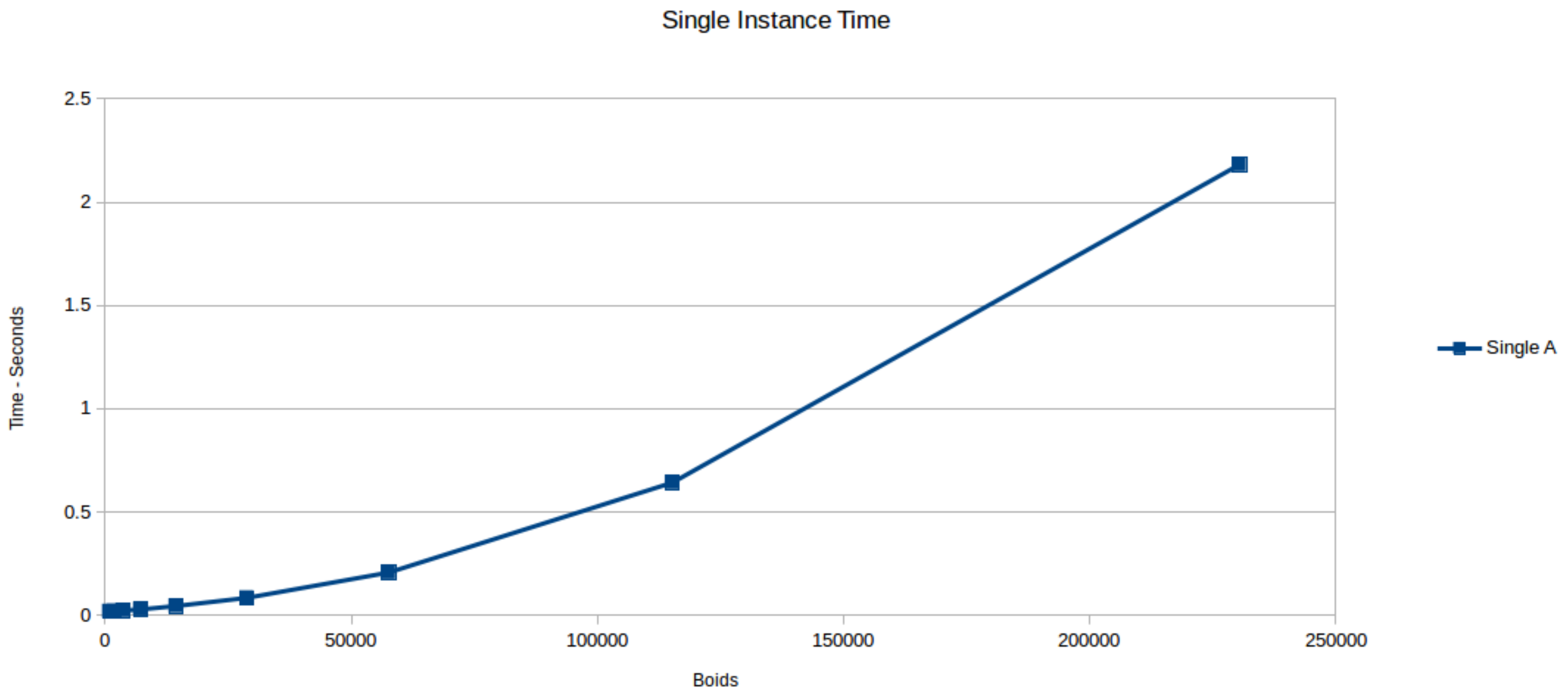
- Running time of algorithm is heavily dependent on Boids configuration
- Tighter flocking results in slower running times
- This implementation:
  - Un-optimized
  - Heavily dependent on arraylists
  - Has a large memory overhead

Test Setup

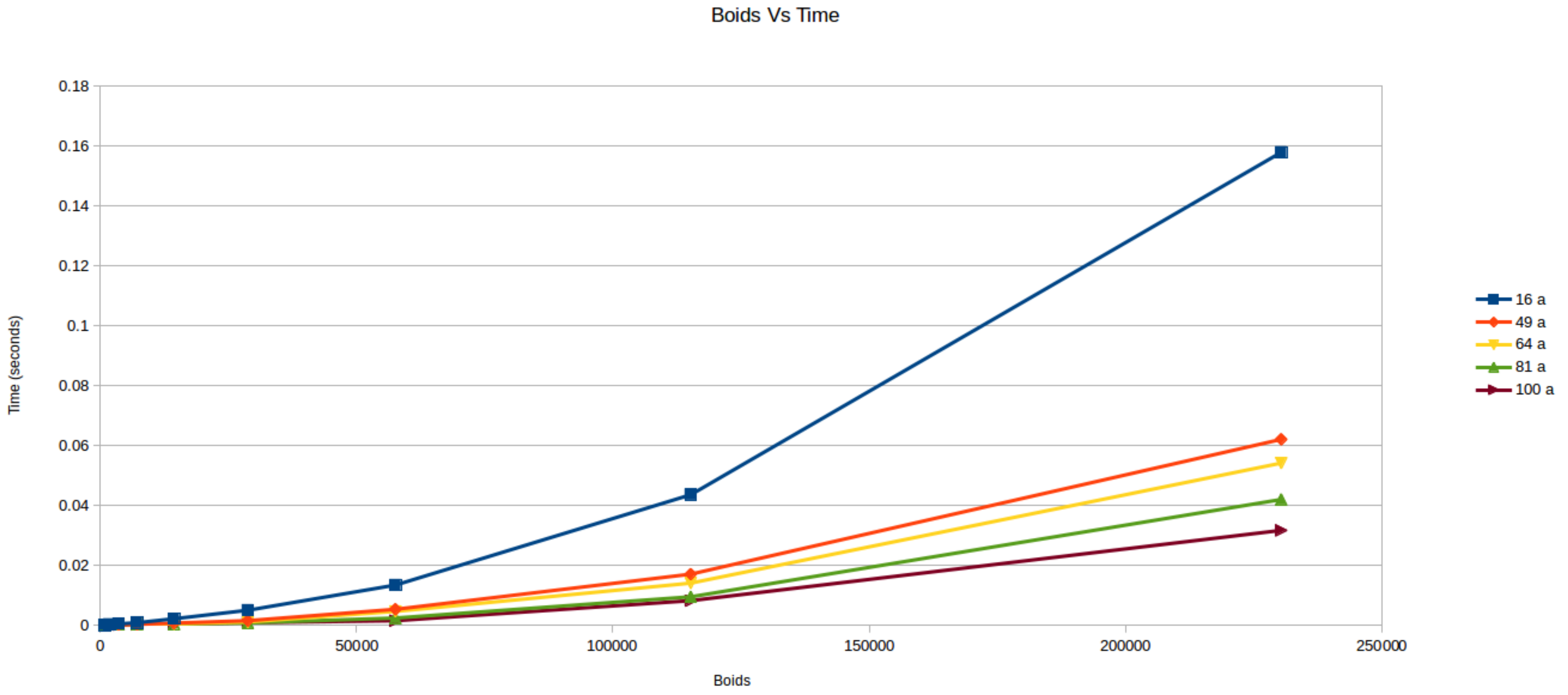
- Fixed area **~4000 x ~4000 m**
- Fixed bin dimension of **40 x 40 m**
- **Variable** bin size per test size
- **Scalable** implementation
- **900 - 230,400** Boids
- **3x3 - 10x10** sub areas



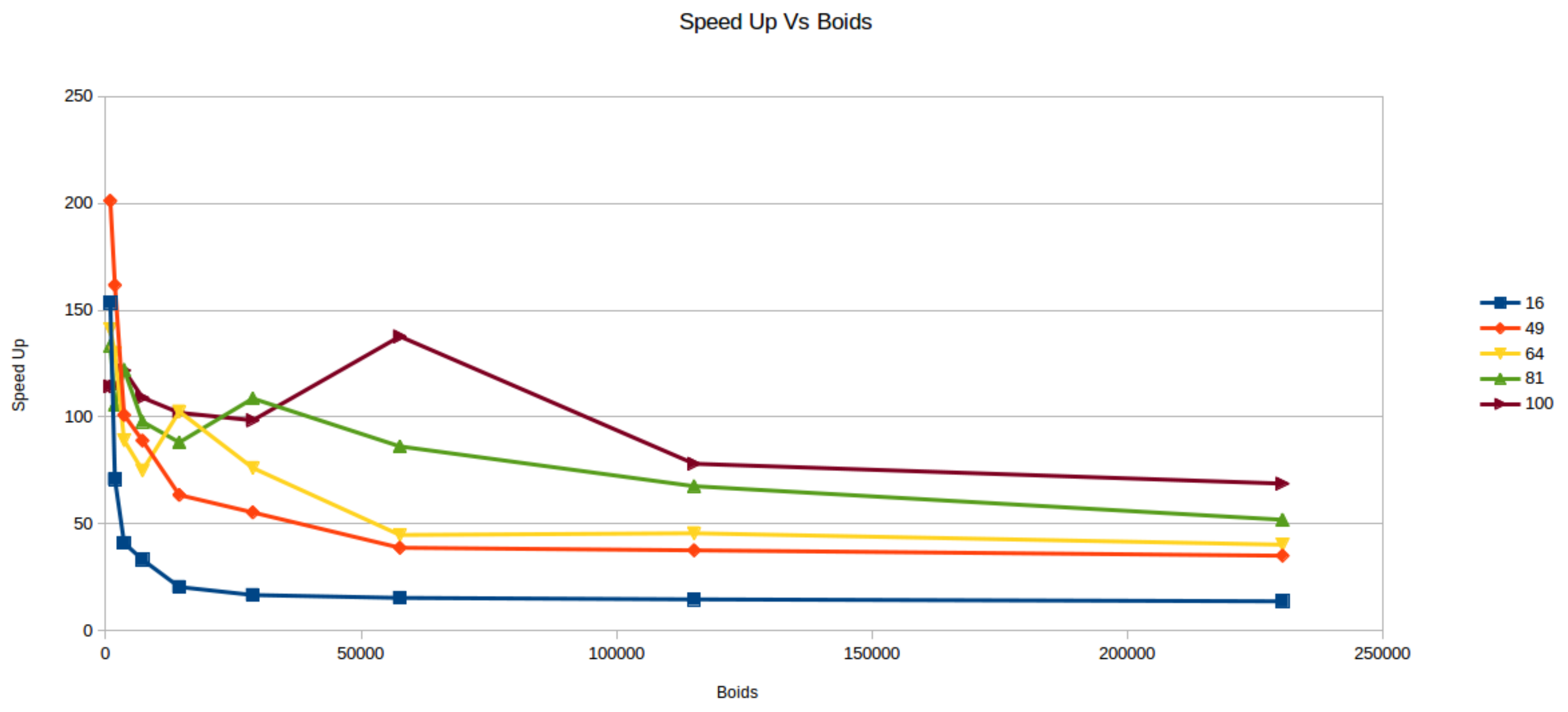
Brute Force Vs Optimized Algorithms



Optimized Algorithm Single Instance



Optimized Algorithm - Parallel Instance Times



Speed Up vs Single Instance

Theoretical Growth Rate

Boids	Comp: Brute	Comp: Grid
900	810000	8100
1800	3240000	16200
3600	12960000	32400
7200	51840000	64800
14400	207360000	129600
28800	829440000	259200

The table above shows the number of comparisons required to perform the algorithm for one cycle for an increasing number of boids

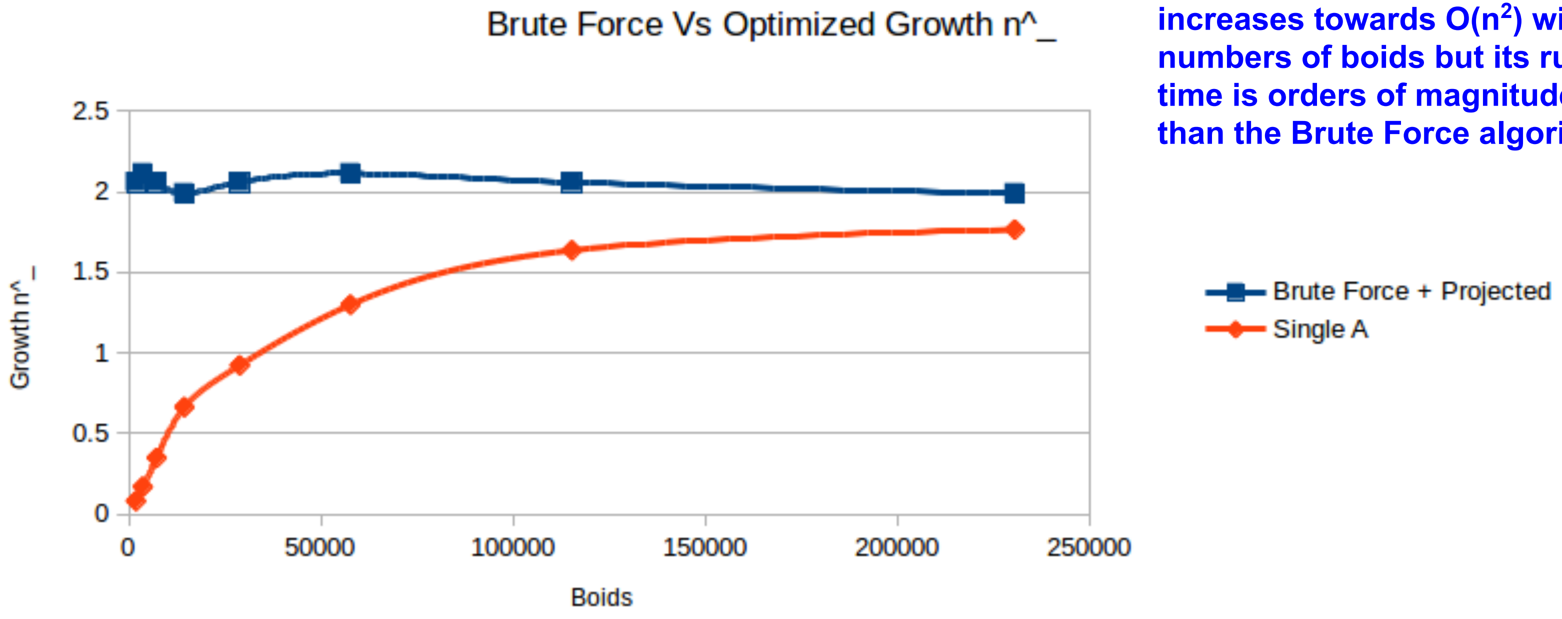
Brute force: (Nested for loop)

As each Boid must be checked against every other Boid to determine if it is in its sphere of influence, this is a O (n^2) algorithm

Grid based: Assuming even Boid distribution

Each Grid has a size that is no less than the sphere of influence of the individual Boids. To correctly perform the simulation for Boid_A, only the Boids in the 8 neighbouring Grids and the Boids in the current Grid need to be checked. As these will be the only Boids that can be in the sphere of influence of Boid_A the simulation will be accurate. With even Boid distribution, this is a O(a*n) algorithm for some constant a

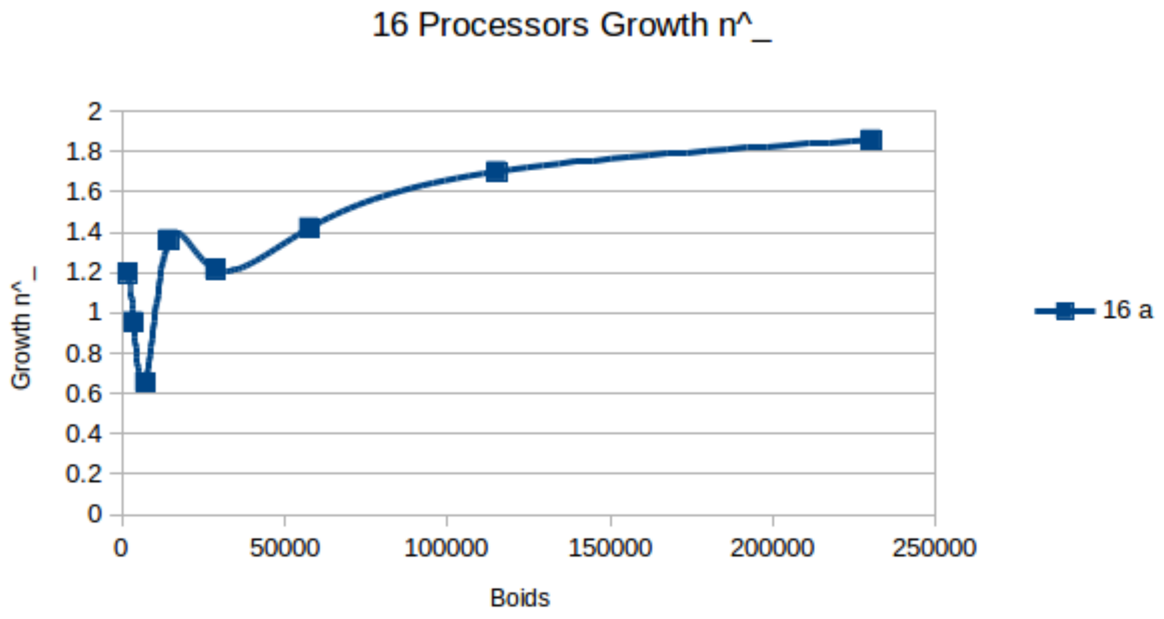
Actual Growth Rate*



- **Growth of the optimized version increases towards O(n^2) with large numbers of boids but its running time is orders of magnitude faster than the Brute Force algorithm**

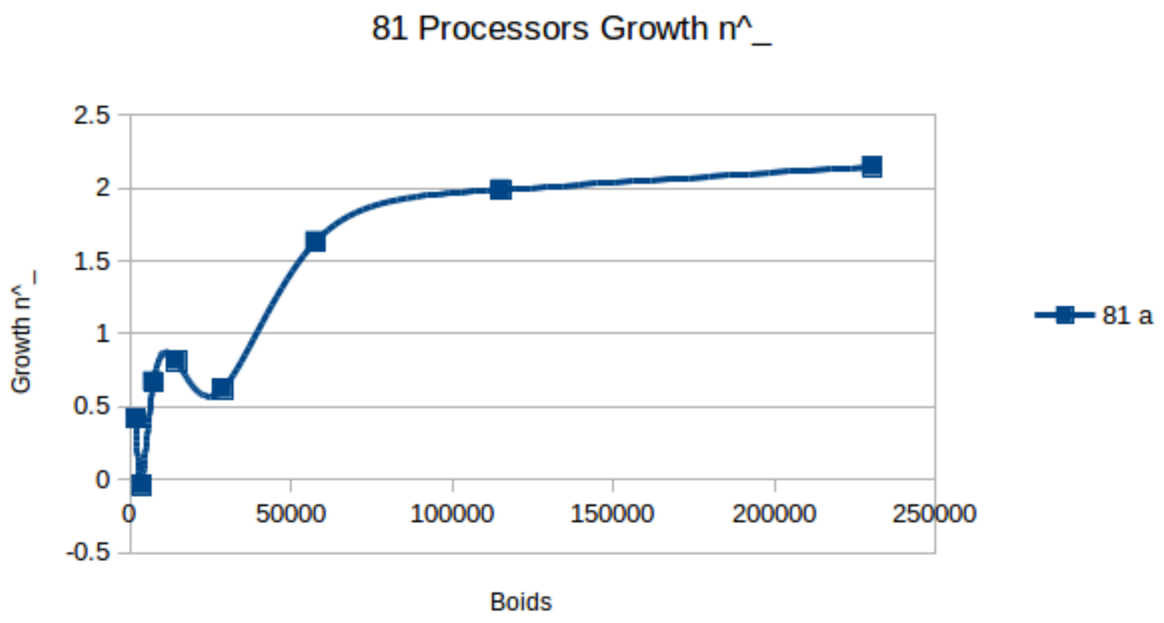
*Running times for brute force are projected above 14400 Boids. The actual growth rate is expected to be at least this amount

Actual Growth Rate 16 proc



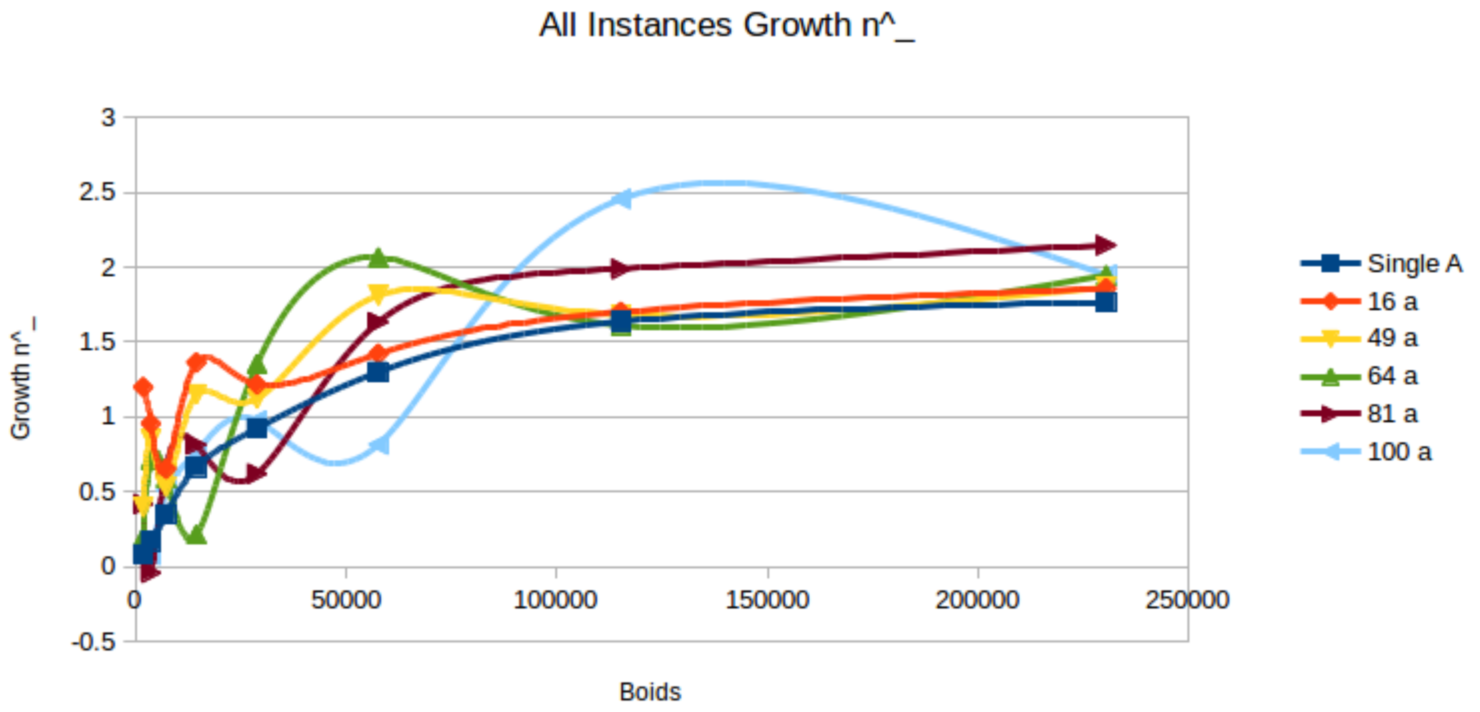
- The growth of the optimized algorithm moves towards O(n^2) after a certain number of boids is reached.
- Note: The tests run with smaller number of Boids will show greater jitter due to the small scale of the simulation, the mpi implementation and certain other random factors inherent to the simulation

Actual Growth Rate 81 proc



- Again, the growth of the optimized algorithm moves towards O(n^2) after a certain number of boids is reached.
- This is true for no matter the number of processors that are used in the test

Actual Growth Rate



Actual Growth Rate

- As the number of Boids increases, the growth rate of the optimized algorithm nears or exceeds $O(n^2)$
- This will be in part due to uneven distribution of the Boids during the simulation for the optimized algorithm. The Boids will naturally flock together in certain regions meaning that for a Boid in a dense region, many more comparisons will be required to correctly execute the simulation.
- As more Boids are added to the same space, the Boids will be more compact compounding the natural grouping nature of the Boids and the growth of the optimized algorithm increases
- For simulations with larger numbers of Boids, even though the growth rate is similar to the brute force algorithm, the actual running time per instance is orders of magnitude faster as can be seen in the first graph above

References

- Arup Guha for sample implementation of arraylist
- Michael Clark <michael@metaparadigm.com> for json-arraylist implementation
- **Flocking** by Daniel Shiffman - Processing Implementation
  - Processing 2.1.1 Example
- Craig Renolds: Original algorithm designer